Medical Image Analysis (2020)

Contents lists available at ScienceDirect

Medical Image Analysis

journal homepage: www.elsevier.com/locate/media

Faster Mean-shift: GPU-accelerated Clustering for Cosine Embedding-based Cell Segmentation and Tracking

Mengyang Zhao[a,1], Aadarsh Jha[b], Quan Liu[b], Bryan A. Millis[c], Anita Mahadevan-Jansen[d], Le Lu[e], Bennett A.Landman[b], Matthew J.Tyska[c], Yuankai Huo[b,*]

[a]*Department of Electrical and Computer Engineering, Tufts University, Medford, MA 02155 USA*
[b]*Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235 USA*
[c]*Department of Cell and Developmental Biology, Vanderbilt University, Nashville, TN 37235 USA*
[d]*Department of Biomedical Engineering, Vanderbilt University, Nashville, TN 37235 USA*
[e]*PAII Inc., Bethesda, MD 20817 USA*

ARTICLE INFO

Article history:
Received xxxxxx
Received in final form xxxxxx
Accepted xxxxxx
Available online xxxxxx

Communicated by xxxxxx

2000 MSC: 41A05, 41A10, 65D05, 65D17

Keywords: Mean-shift, GPU, Cell Tracking, Cell Segmentation, Embedding

ABSTRACT

Recently, single-stage embedding based deep learning algorithms gain increasing attention in cell segmentation and tracking. Compared with the traditional "segment-then-associate" two-stage approach, a single-stage algorithm not only simultaneously achieves consistent instance cell segmentation and tracking but also gains superior performance when distinguishing ambiguous pixels on boundaries and overlaps. However, the deployment of an embedding based algorithm is restricted by slow inference speed (e.g., ≈1-2 mins per frame). In this study, we propose a novel Faster Mean-shift algorithm, which tackles the computational bottleneck of embedding based cell segmentation and tracking. Different from previous GPU-accelerated fast mean-shift algorithms, a new online seed optimization policy (OSOP) is introduced to adaptively determine the minimal number of seeds, accelerate computation, and save GPU memory. With both embedding simulation and empirical validation via the four cohorts from the ISBI cell tracking challenge, the proposed Faster Mean-shift algorithm achieved 7-10 times speedup compared to the state-of-the-art embedding based cell instance segmentation and tracking algorithm. Our Faster Mean-shift algorithm also achieved the highest computational speed compared to other GPU benchmarks with optimized memory consumption. The Faster Mean-shift is a plug-and-play model, which can be employed on other pixel embedding based clustering inference for medical image analysis. (Plug-and-play model is publicly available: https://github.com/masqm/Faster-Mean-Shift)

1. Introduction

With technical evolution in microscopy imaging, biomedical research has been advanced by spatial-temporal cell imaging, to understand cell motility and cell proliferation (Zimmer et al., 2006), embryonic development (Montell, 2008), tumorigenesis (Condeelis and Pollard, 2006) etc. To obtain dynamic information from acquired cell images and videos, manual tracing regarded as the gold standard of quantifying spatial-temporal microscope images. However, such a process is not only laborious and tedious but also impractical when terabyte (TB) level imaging data are acquired per day from a single imaging center (Rosenthal et al., 2010). Therefore, automatic cell segmen-

*Corresponding author. Email: yuankai.huo@vanderbilt.edu
[1]First author. Email: Mengyang.Zhao@tufts.edu

tation and tracking are crucial in cell image and video analyses, especially with the increasing spatial and temporal resolution from modern microscopy imaging.

Many computer-assisted segmenting and tracking methods (Webb et al., 1996; Allen et al., 1998; Czirók et al., 1998) have been proposed over the past decades. Recently, with the explosive growth of artificial intelligence (AI) and deep learning technologies (e.g., convolutional neural network (CNN), and the long short-term memory model (LSTM) (LeCun et al., 1998; Hochreiter and Schmidhuber, 1997)), the performance of automatic cell segmentation and tracking has also been leveraged dramatically (Debeir et al., 2005; Bulte, 2009; Sutton et al., 2008; Ronneberger et al., 2015; Chen et al., 2017; Graham et al., 2019). Such methods were typically implemented as a "segment-then-track" two-stage paradigm, which linked segmented cells across frames via association (e.g., bipartite graph matching) algorithms. The advantages of such methods are (1) a clear problem definition, and (2) scalable to high resolution cell images. However, the overall performance depends on both segmentation and association as independent tasks, with-out integrating the two synergetic tasks simultaneously.

To aggregate the synergetic tasks as a holistic single stage algorithm, Payer et al. (Payer et al., 2019) proposed a cosine embedding based recurrent stacked hourglass network (RSHN) for instance cell segmentation and tracking using microscope video sequences. They approached the instance segmentation and tracking problem from a new pixel-wise cosine embedding perspective to maximize the embedding similarity of the pixels within the same cell, while minimizing the embedding similarity across different cells. This method approached both the instance segmentation and tracking of cells within a single uniformed framework, which achieved superior performance compared with previous two-stage cell image processing approaches.

However, the major limitation of (Payer et al., 2019), which is a common issue in pixel embedding methods, is the slow inference speed when applying the trained model on testing images. For instance, the time for processing a single frame from the ISBI challenge data-set can take nearly two minutes per frame, which is a major bottleneck of deploying such algorithms on large-scale data. Based on our analyses, the majority of the computational time in (Payer et al., 2019) was spent on the clustering of embedding voxel-level features, as the time complexity of this algorithm is $O(Tn^2)$, where T is the time for processing each pixel point and n is the number of pixel points. Although there are several methods, such as adopting KD tree (Xiao and Liu, 2010) or ball-tree (Chavez, 2001), could be employed to accelerate the algorithm. However, considering the large number of pixels in the image, the computational burden is much heavier than the 1D data in canonical clustering tasks, which leads to the considerably slow inference for medical image analysis.

Mean-shift is arguably the most widely used clustering algorithm in a large number of embedding based image processing, which is able to determine the number of clusters adaptively, as opposed to other clustering approaches (e.g., k-means (Senoussaoui et al., 2013)) with a fixed number of clusters. In cell image processing, the mean-shift algorithm is proven to be more accurate (5-10%,) than other clustering algorithms (Payer et al., 2019; Campello et al., 2015). To accelerate the speed of mean-shift clustering, GPU accelerated algorithms with parallel computing have been proposed. For instance, the fast mean-shift algorithm (Miaoqing et al., 2013) was developed to achieve significant speed-up compared with CPU based mean-shift clustering. Recently, (Fang et al., 2019) further accelerated computational speed with parallel tensor operations has been achieved. However, (Fang et al., 2019) is memory extensive, which is infeasible for processing high resolution image frames (e.g., 512×512 in Figure 1) using the ordinary GPU cards. Therefore, a faster GPU accelerated clustering method with reasonable GPU memory consumption, is imperative for embedding based medical image analysis,

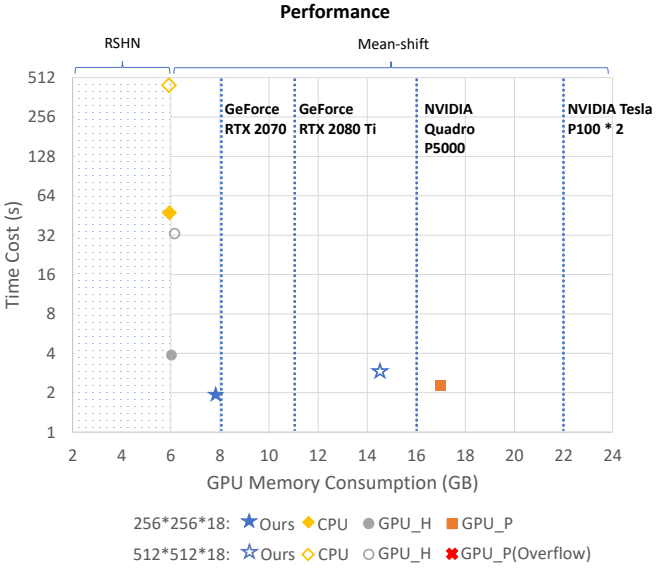


Fig. 1: This figure shows the overall computational speed and GPU memory consumption when performing mean-shift clustering on simulated tensors with 256×256×18 and 512×512×18 resolution. The 256×256 and 512×512 indicates the image resolution for each frame, while 18 represents the dimensions of feature vectors of each embedded pixel. "CPU" represents the performance of (Payer et al., 2019), which executed mean-shift only using CPU. "GPU_H" (Miaoqing et al., 2013) and "GPU_P" (Fang et al., 2019) are two previously proposed GPU-accelerated fast mean-shift algorithms, where "H" and "P" indicate "hybrid" and "parallel" GPU accelerations respectively. The details are presented in Section 4.

Inspired by a parallel k-means algorithm (Bhimani et al., 2015), we believe that it is not necessary to compute feature vectors for all pixels when dealing with embedding based clustering. In this paper, we propose a novel GPU tensor accelerated mean-shift clustering algorithm, called Faster Mean-shift, to speed up the recurrent neural network (RNN) based cosine embedding framework for holistic cell instance segmentation and tracking. To optimize the GPU memory consumption, the online seed optimization policy (OSOP) and early stopping strategy are proposed to reduce unnecessary computing. The simulation as well as four real cohorts from the ISBI cell tracking challenge (Ulman et al., 2017) are employed in this study, to evaluate the accuracy, time cost, and GPU memory consumption of the proposed and baseline algorithms. We integrated the

proposed Faster Mean-shift algorithm into the state-of-the-art RSHN framework (Payer et al., 2019), to achieve 7-10 times speed-up during the inference stage, without sacrificing accuracy. The testing results show that our algorithms achieved the best computational speed with optimized memory consumption (Figure 1).

In summary, the main contributions of this study are as follows:

(1) We propose a novel Faster Mean-shift algorithm, which accelerates the embedding clustering based one-stage holistic cell instance segmentation and tracking.

(2) We propose the new online seed optimization policy (OSOP) and early stopping strategy to achieve the best computational speed with optimized memory consumption, compared with previous GPU-based benchmarks. (Fang et al., 2019; Miaoqing et al., 2013).

(3) The proposed Faster Mean-shift achieved 7-10 times speed-up compared with the state-of-the-art embedding based cell instance segmentation and tracking algorithm (Payer et al., 2019).

(4) Comprehensive simulations for in-depth theoretical analysis, as well as empirical validations on four ISBI cell tracking challenge data-set (Ulman et al., 2017) have been performed in this study.

The rest of the paper is organized as follows. In Section 2, we introduce background and related research relevant to cell segmentation and tracking. In Section 3, the mean-shift clustering and our proposed acceleration methods are presented. It includes the Faster Mean-shift algorithm and the theoretical derivation of OSOP. Section 4 focuses on presenting the implementation details and experimental results. Then, in Section 5 and 6, we provide ablation studies and conclude our work.

2. Related Research

Our research is closely related to holistic instance segmentation and tracking (Payer et al., 2019), as well as GPU accelerated clustering algorithms (Fang et al., 2019; Miaoqing et al., 2013). We present a brief introduction to the related research in this section.

2.1. Instance Image Segmentation and Tracking

Fully automatic image instance segmentation and tracking plays a critical role in biomedical image analyses, such as quantifying cells, cell nuclei, and other sub-millimeter structures from microscope images. Currently, the de facto standard instance segmentation and tracking methods are based on a deep learning algorithms (Ulman et al., 2017).

Deep learning can be traced back to 1998 when LeNet-5 (LeCun et al., 1998), a basic CNN model, was proposed by LeCun et al., where the convolutional layers were proposed to extract more generalizable image features than traditional feature engineering (Zhao et al., 2020). Then, a series of improvements, such as the ReLU activation function in AlexNet (Krizhevsky et al., 2012) and the region selection algorithm in R-CNN (Girshick et al., 2014), greatly improved the network recognition performance for processing the real-world image data. Different from processing instant image data, recurrent neural networks such as the LSTM (Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU) (Cho et al., 2014) are proposed to incorporate temporal information into the deep learning framework.

Among different deep learning algorithms, the embedding based cell instance segmentation and tracking approach (Payer et al., 2019) is the most related ones to our study. Meanwhile, without doing tracking, some other works focused on embedding based instance cell segmentation (Kulikov and Lempitsky, 2020; Chen et al., 2019). For embedding based methods, one major computational bottleneck is to cluster pixel-wise feature vectors to generate the final instances.

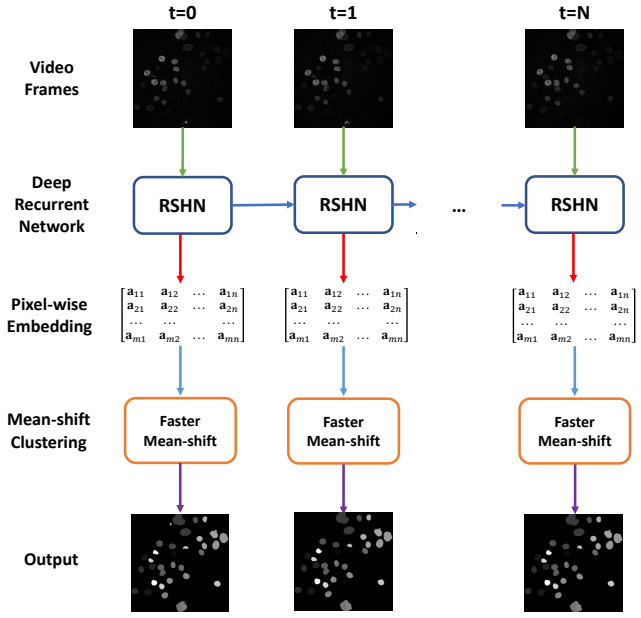


Fig. 2: This figure presents the overall framework of a pixel embedding based holistic instance segmentation and tracking algorithm, with the proposed plug-and-play Faster Mean-shift GPU accelerated clustering algorithm.

2.2. Single-stage Solution with Cosine Embedding

Recently, instance cell segmentation and tracking have received increasing attention in many public challenges, such as the ISBI cell tracking challenge (Ulman et al., 2017). Traditionally, instance segmentation and object tracking were performed separately as a two-stage design. In 2019, (Payer et al., 2019) integrated the instance segmentation and cell tracking into a holistic stacked hourglass network (Newell et al., 2016) with pixel embedding based design. The well-known mean-shift clustering algorithm (Comaniciu and Meer, 2002) was employed during the inference stage to achieve final cell instance segmentation and tracking results.

The RSHN model (Payer et al., 2019) was developed from the convolutional GRUs (ConvGRUs) (Ballas et al., 2015), and the stacked hourglass network (Newell et al., 2016). It used a ConvGRU with 3×3 filters and 64 outputs between the temporal paths to represent states and stacked two hourglasses in a row to improve network predictions. Based on the RSHN

framework, each pixel in a video frame was converted to a high-dimensional embedding vector. The embedding vectors from different cell instances were distinguished by cosine similarity (Li et al., 2004). For example, **A** was the embedding vector of pixel a, and **B** was the embedding vector of pixel b. The cosine similarity of **A** and **B** is defined as:

$$cos\,(\mathbf{A},\mathbf{B}) = \frac{\mathbf{A}\cdot\mathbf{B}}{||\mathbf{A}||\ ||\mathbf{B}||} \tag{1}$$

which ranged from -1 to 1, where 1 indicates that two vectors have the same direction, 0 indicates orthogonal, and -1 indicates the opposite. Then, if $cos\,(\mathbf{A},\mathbf{B})$ was $\approx$ 1, pixel a and pixel b were likely from the same instance. $\approx$ 0 was likely from different instances. After converting a video frame to an embedding vector matrix, a clustering algorithm was used to classify the pixels into different clusters/instances. According to (Payer et al., 2019), the results obtained by the mean-shift algorithm were better than other clustering algorithms (Campello et al., 2015). The tracking and segmentation accuracy was improved 5-10%, which demonstrated the advantages of the mean-shift algorithm.

2.3. Fast Mean-shift Algorithms

Mean-shift is an iterative clustering algorithm, which determines the number of clusters adaptively, as opposed to other clustering approaches (e.g., k-means (Senoussaoui et al., 2013)). However, the major limitation of the mean-shift algorithm is the intensive computation, with a time complexity of $O(Tn^2)$. T is the number of iterations for processing each data point and n is the total number of data points in the data-set (Comaniciu and Meer, 2002). Thus, due to the large number of pixels in image processing tasks, utilizing mean-shift clustering requires a large amount of computational time.

Several mean-shift acceleration methods have been developed to accelerate the mean-shift algorithm. These methods are generally divided into two families: CPU acceleration and GPU acceleration. For CPU acceleration, one major improvement is to reduce the number of vectors required in the mean-shift vector calculation. Traditionally, the calculation of the mean-shift requires all vectors from all the pixels. Therefore, the computational speed is accelerated if the algorithm is performed on a sub-sample of the feature vectors. To achieve better vector management, the fast searching algorithm proposed by Chalela et al. (Chalela et al., 2019) introduced a grid based searching paradigm. Another work (Xiao and Liu, 2010) used the KD tree to manage the vectors and further accelerate the searching approach. Based on the KD tree (Xiao and Liu, 2010) and ball-tree (Chavez, 2001), sklearn (sklearn, 2015) provided a CPU accelerated mean-shift clustering implementation.

Although the aforementioned searching algorithms greatly improve speed, the overall processing speed of the CPU version is still not satisfactory for high dimensional features. For instance, the time for processing a 256×256 pixels image could take more than one minute, and even longer for a higher resolution image. Therefore, GPU accelerated solutions were proposed to further speed up the mean-shift. Huang et al. (Miaoqing et al., 2013) proposed a hybrid CPU/GPU acceleration for mean-shift clustering. In their algorithm, GPU was used for calculating the mean-shift vector. However, because of the high time complexity of the region fusion step in their algorithm, the speedup for large image clustering yields inferior performance, at only 1.9 times. Huang et al (Fang et al., 2019) proposed a fully parallel mean-shift algorithm. Instead of only using a GPU for vector calculation, in their algorithm, all vectors were shifted at the same time. This complete parallelization greatly sped up the algorithm. In the best case, their mean-shift clustering was approximately 40 times faster than the CPU version. However, this algorithm is limited by intensive resource consumption, such as GPU memory and CUDA cores, to complete parallel computing. Therefore, in their paper, large GPU clusters(with 64 worker GPUs and 384 GB GPU memory) were required to achieve the best acceleration. Thus, the consumption of computing resources could be unbearable for normal single GPU scenarios.

Indeed, in our implementation of (Fang et al., 2019), the memory resources of a single GPU card (12 GB GPU Memory) are quickly exhausted for the video embedding with 18 features and 512×512 image resolution. Therefore, it is necessary to develop a new acceleration method, which has memory consumption that is acceptable for normal GPU cards, with even faster computational speed. Inspired by a parallel k-means algorithm (Bhimani et al., 2015), we realized that it is not necessary to calculate all vectors when dealing with clustering problems. Therefore, unlike Huang et al. (Fang et al., 2019), who parallelized all the vectors, we propose a Faster Mean-shift clustering algorithm based on adaptively determining a subset of vectors for computing.

3. Proposed Method

Our proposed Faster Mean-shift method is presented in this section. First, we introduce the mean-shift clustering for cosine embedding. Then, we present the proposed Faster Mean-shift algorithm with detailed theories and implementations.

3.1. Mean-shift Clustering for Cosine Embedding

The mean-shift algorithm is one of the most popular vector-based clustering methods, which is unsupervised and training-free (Senoussaoui et al., 2013). Assume there is a given vector $\mathbf{x}_g$ in a vector set $S = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ of unlabeled data. The standard form of the estimated kernel density function $\hat{f}(x)$ at $\mathbf{x}_g$ is given by the following formula:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} k\left(\frac{d(\mathbf{x},\mathbf{x}_i)}{h}\right) \tag{2}$$

where $k(\mathbf{x})$ is a kernel function, $d(\mathbf{x})$ refers to the distance function, and h is referred to as the kernel bandwidth.

A standard kernel function is the Epanechnikov kernel (Guo et al., 2007) given by the following formula:

$$k(\mathbf{x}) = \begin{cases} 1 & ||\mathbf{x}|| \le 1 \\ 0 & ||\mathbf{x}|| > 1 \end{cases} \tag{3}$$

The Euclidean distance function between two vectors is:

$$d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\| \tag{4}$$

In the mean-shift clustering algorithm, the mean-shift vector is derived by calculating the gradient of the density function.

$$\begin{aligned}\nabla \hat{f}(\mathbf{x}) &= \frac{1}{nh^d}\sum_{i=1}^{n} k\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right) \\ &= \frac{1}{nh^d}\sum_{i=1}^{n} \nabla k\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right) \\ &= \frac{2}{nh^{d+2}}\left[\sum_{i=1}^{n} k\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right)\right] \\ &\times \left[\frac{\sum_{i=1}^{n} \mathbf{x}_i k\left(\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right)^2\right)}{\sum_{i=1}^{n} k\left(\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right)^2\right)} - \mathbf{x}\right]\end{aligned} \tag{5}$$

Then, the content within the "[·]" in the Eq.(5) is the mean-shift vector, presented as the following expression:

$$M_h(\mathbf{x}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i k\left(\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right)^2\right)}{\sum_{i=1}^{n} k\left(\left(\frac{d(\mathbf{x}, \mathbf{x}_i)}{h}\right)^2\right)} - \mathbf{x} \tag{6}$$

For a subset of feature vectors satisfying $S_h = \{\mathbf{x}_i | d(\mathbf{x}_g, \mathbf{x}_i) \leq h, \mathbf{x}_i \in S\}$, we reform the mean-shift vector as:

$$M_h(\mathbf{x}) = m_h(\mathbf{x}) - \mathbf{x} \tag{7}$$

where the sample mean $m_h(\mathbf{x})$ is defined as:

$$m_h(\mathbf{x}) = \frac{1}{m}\sum_{\mathbf{x}_i \in S_h} \mathbf{x}_i \tag{8}$$

The iterative processing of calculating the sample mean converges the data to modes, which are the predicted clustering patterns. The proof of its mathematical convergence is provided in (Comaniciu and Meer, 2002; Fukunaga and Hostetler, 1975). The iterative process of mean-shift clustering is depicted as Algorithm 1

It is worth mentioning that, in this paper, we mainly use the cosine distance function in mean-shift, where the equation (4) needs to be replaced with the cosine distance between two vectors:

$$d(\mathbf{x}_1, \mathbf{x}_2) = 1 - \left(\frac{\mathbf{x}_1 \cdot \mathbf{x}_2}{\|\mathbf{x}_1\| \, \|\mathbf{x}_2\|}\right) \tag{9}$$

By using mean-shift clustering, the output cosine embedding vectors from the RSHN (Payer et al., 2019) model are clustered into different instances. According to the experiments in paper (Payer et al., 2019), the combination of mean-shift clustering and RSHN provided accurate instance segmentation and tracking results. However, the efficiency of such a method is limited by the intensive time consumption when performing standard mean-shift clustering (see Section 4).

Algorithm 1 Mean-shift Clustering

Require: h: Bandwidth; S: Vector set;
Ensure: *modes*: The modes of each cluster;
1: **for** $\mathbf{x} \in S$ **do**
2: # Initialization for each vector
3: $\mathbf{x}_g \leftarrow \mathbf{x}$
4: Create a window: Bandwidth: h, Center: $\mathbf{x}_g$
5: # Mean-shift iteration
6: **while** $\mathbf{x}_g$ not converge **do**
7: $\mathbf{x}_g \leftarrow m_h(\mathbf{x}_g)$
8: Update window to the new center
9: **end while**
10: *modes* append $\mathbf{x}_g$
11: **end for**
12: Prune *modes*

3.2. Faster Mean-shift Algorithm

3.2.1. GPU-based parallelization

Inspired by mean-shift GPU parallelization (Fang et al., 2019) and k-means parallel implementation (Bhimani et al., 2015), we propose Faster Mean-shift algorithm, a novel GPU accelerated parallel mean-shift algorithm. The core idea of our proposed algorithm is to adaptively determine the number of seeds with an early stopping strategy to reduce the number of iterations in the mean-shift computation. The pseudo-code of the algorithm is shown in Algorithm 2.

Algorithm 2 Faster Mean-shift Clustering

Require: h: Bandwidth; S: Vector set;
Ensure: $\mathbf{x} - modes$: A vector-modes list
1: # Seed Selection
2: Evenly random select seed vector set $S_{seed} \in S$
3: # Parallelization with GPU
4: **for** $\mathbf{x}_{seed} \in S_{seed}$ **do**
5: **while** $\mathbf{x}_{seed}$ not converge **do**
6: $\mathbf{x}_{seed} \leftarrow m(\mathbf{x}_{seed}) \cdot k_h(\mathbf{x}_{seed}, \mathbf{x}_i)$
7: **end while**
8: *modes* append $\mathbf{x}_{seed}$
9: **end for**
10: Prune *modes*
11: **for** $\mathbf{x} \in S$ **do**
12: Cluster $\mathbf{x}$ by the distance to *modes*
13: **end for**

The algorithm first selects a batch of seed vectors from the input vector set S. According to our settings, in general, N vectors are randomly selected from S to form a subset S_{seed}. Then, these batched seeds are pushed into the GPU to perform parallel computation in lines 4-9 in Algorithm 2. The mean-shift iterations are performed for each seed vector simultaneously. To save communication time on the GPU side, our algorithm does not search which points belong to the S_h set. Instead, shown on line 6 in Algorithm 2, our algorithm uses $m(\mathbf{x})$ to calculate the mean-shift vector with all other points and then multiplies it with the kernel function, $k_h(\mathbf{x}, \mathbf{x}_i)$, to obtain the mean-shift vector. The $m(\mathbf{x})$ and $k_h(\mathbf{x}, \mathbf{x}_i)$ functions are given by the following

formula:

$$m(\mathbf{x}) = \frac{1}{m} \sum_{\mathbf{x}_i \in S} d(\mathbf{x}, \mathbf{x}_i) \tag{10}$$

$$k_h(\mathbf{x}, \mathbf{x}_i) = \begin{cases} 1 & d(\mathbf{x}, \mathbf{x}_i) \leq h \\ 0 & d(\mathbf{x}, \mathbf{x}_i) > h \end{cases} \tag{11}$$

Next, the position of $\mathbf{x}_i$ is updated according to the mean-shift vector. If the change of the distance between two iterations is less than a threshold (typically $h/1,000$), then the computation for this seed vector $\mathbf{x}_{seed}$ has converged. After parallelly manipulating the batch of seed vectors in the GPU, the modes are obtained in the vector set S. Next, such modes are further pruned and merged if their distance is small. In the end, all vectors are clustered according to their distance to each mode to obtain the final result.

Since only the seed vectors need to be manipulated in parallel, the GPU memory consumption is dramatically reduced, which enables parallel computing for mean-shift clustering on only one GPU card. However, to segment the real data-set, two critical tasks need to be tackled: (1) to determine and adjust the number of seed vectors, and (2) to reduce seed convergence time.

3.2.2. Online Seed Optimization Policy (OSOP)

The number of seed vectors plays an important role in the mean-shift algorithm. If the number of seeds is too low, the clustering results might not cover all modes. On the other hand, too many seeds lead to large GPU memory consumption (Fang et al., 2019).

As opposed to traditional methods, wherein number of seeds are fixed, we proposed the OSOP approach to determine the minimum number of seeds adaptively, based on the number of instances and the foreground area in each prediction. For cell instance segmentation, we hypothesize that (1) the sizes and spatial distributions of the cells are homogeneous, and (2) the area ratio between the foreground $A_{foreground}$ and entire image A_{image} is r. Based on such a hypothesis, the distribution of a seed follows a binomial distribution, where the percentage of each instance $A_{foreground}$ is presented as:

$$A_{foreground} = A_{\text{instance}} \times I = A_{\text{image}} \times r \tag{12}$$

where I is the number of cell instances and A_{instance} is the average area of cells. Therefore, the probability that one random seed located within a particular instance cluster P_{seed} is

$$P_{seed} = \frac{A_{\text{instance}}}{A_{\text{image}}} = \frac{r}{I} \tag{13}$$

From the probabilistic model, the probability of a seed being located in an instance is a Binomial distribution. As a result, if the total number of seeds is N, the probability that each cluster has at least one seed is:

$$P_{seed/cluster} = \left(1 - \left(1 - \frac{r}{I}\right)^N\right)^I \tag{14}$$

However, the hypothesis of an equal size of the instances in Eq. (13) does not hold in the real world. Therefore, the probability P of each cluster having at least one seed is lower than the upper bound:

$$P \leq \left(1 - \left(1 - \frac{1}{2I}\right)^N\right)^I \tag{15}$$

$$\ln\left(1 - \left(1 - \frac{1}{2I}\right)^N\right) \geq \frac{\ln(P)}{I} \tag{16}$$

$$\left(1 - \frac{1}{2I}\right)^N \leq 1 - e^{\frac{\ln(P)}{I}} \tag{17}$$

$$N \ln\left(1 - \frac{1}{2I}\right) \leq \ln\left(1 - e^{\frac{\ln(P)}{I}}\right) \tag{18}$$

Therefore, for a desired P, the N should satisfy:

$$N \geq \frac{\ln\left(1 - e^{\frac{\ln(P)}{I}}\right)}{\ln\left(1 - \frac{r}{I}\right)} = N_{\min} \tag{19}$$

In mean-shift clustering, each instance/cluster should have at least one seed to ensure all instances are successfully recognized. Therefore, if we expect the probability P that the seed number would be sufficient for the I instance above (e.g., $\geq 99\%$), we should have at least N seeds above the low bound $N_{\min}$, where the I and r are calculated from the predicted segmentation.

In our algorithmic implementation, since the areas of all cell instances are not identical, we typically need a larger number of seeds to cover all instances. A constant coefficient $\alpha \geq 1$ is introduced to enlarge the minimal numbers of seed in our algorithm.

$$N \geq \alpha \cdot N_{\min} \tag{20}$$

3.2.3. Early Stopping

To further accelerate computational speed, we propose the early stopping strategy to optimize the total computational time for all the seed vectors. In an ideal situation, all seed vectors would converge simultaneously. However, the convergence is heterogeneous, where a few seeds will converge considerably slow and become the bottleneck of the entire GPU computing. To tackle this issue, we set a threshold percentage of converged seeds as γ. If more than γ percentage of the seed vectors are converged, the mean-shift optimization of such iterations are terminated. The seed vectors that fail to convergence are discarded from the seed vector. In all experiments in this study, we empirically set the $\gamma = 90\%$.

Consider Eq. (20), the required minimal number of seeds for each iteration as:

$$N \geq \frac{\alpha}{\gamma} \cdot N_{\min} = L \cdot N_{\min} \tag{21}$$

where $L \cdot N_{\min}$ is the minimal seed numbers for Faster Mean-shift implementation.

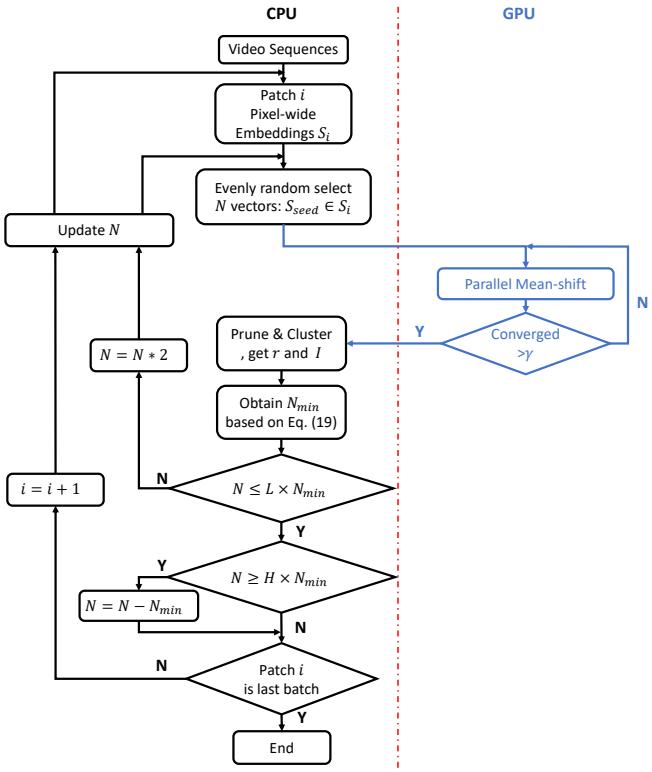


Fig. 3: This figure depicts the flowchart of Faster Mean-shift. The black portions of the diagram are processed by CPU, while the blue portions are processed by GPU.

3.2.4. *Faster Mean-shift Clustering.*

The flowchart of the entire Faster Mean-shift algorithm is shown in Figure 3. The computation is divided into two parts: the GPU and the CPU. The GPU portion mainly executes the iterative mean-shift computation in parallel (lines 4-9 in Algorithm 2), while the CPU portion controls the number of seeds.

The CPU is mainly responsible for adjusting the number of seed vectors. Initially, N is set to 128 ($N_{initial} = 128$). Then, during the following iterations, $N_{\min}$ will be updated based on the observed I and r in Eq. (19). If N is less than L times of $N_{\min}$, the N will be doubled for the next iteration. If N is larger than H times of the $N_{\min}$, the computational cost would be too expensive. In that case, N is reduced as $N - N_{\min}$ for the next iteration. Based on our simulations, we empirically set $L = 2$ and $H = 8$ for all studies.

4. Experiments and Results

In this study, we performed both simulation and empirical validations via the ISBI cell tracking challenge data-set to evaluate the performance of the proposed Faster Mean-shift algorithm.

4.1. Environments

This research uses a standard_NC6 (Microsoft, 2020) virtual machine platform at the Microsoft Azure cloud. The virtual machine includes one-half NVIDIA Tesla K80 accelerator (NVIDIA, 2015) card and six Intel Xeon E5-2690 v3 (Haswell) processors. The K80 GPU with 12 GB GPU memory was used in this study. The memory of the standard_NC6 (Microsoft, 2020) virtual machine was 56 GB. The Faster Mean-shift algorithm was implemented with PyTorch and Python3. The source code of our proposed method has been publicly available [2]. To allow for a fair comparison on the same platform, we re-implemented the GPU accelerated mean-shift baselines (Fang et al., 2019; Miaoqing et al., 2013) using PyTorch and Python3, which were originally implemented using the C language and OpenGL. The RSHN algorithm and CPU version of mean-shift were obtained from (Payer et al., 2019). During training, the learning rate was initially set to 0.0001, and decreases to 0.00001 after 10,000 iterations.

4.2. Simulation

The purpose of the simulation experiment is to evaluate the performance of the proposed Faster Mean-shift across different distributions and measure the costs of computational time and GPU memory.

4.2.1. Data

Since the cosine similarity is used to distinguish vectors, we used polar coordinates to present the three distributions of simulated data as three rows in Figure 4. In the circle distribution, 1,500 data points with embedding dimension = 2 were generated as two concentric circles with different radii. All data points were evenly distributed into 10 clusters. Gaussian noise with a standard deviation of 0.02 was added to the data. The second and third rows in Figure 4 indicated polarized distributions with eight and four hidden clusters. Gaussian noise was added with a standard deviation of 0.005 (for the data with seven instances) and 0.01 (for the data with three instances). In the circle distribution (first row in Fig. 4), the intensities of Gaussian noise were evenly distributed from 0.95 to 1.05 (large circle) and 0.475 to 0.525 (small circle) on the polar-diameter directions, whose variations were determined by 10% of the circles' radius. In the simulations with three and seven instances (second and third rows in Fig. 4), the intensity is evenly distributed from 1 to 2. Different from even distribution in the first simulation, we simulated the proportion of the background in the real images, where the blue data-set had 750 points, accounting for 50% of the total points. The remaining points were divided evenly across different clusters.

4.2.2. Design

We generated eight sets of simulation data with sizes of 1K, 2K, 5K, 10K, 20K, 50K, 100K, and 200K (1K=1,000), where each data point had 18 dimensions from ten randomly distributed clusters using *blobs* function. For each set, the cluster number was set to 10, with a Gaussian noise that had a standard deviation of 0.01. The vectors were also orthogonal or opposite in different clusters to ensure distinct cosine similarities. We compared our algorithms with the CPU version of mean-shift (CPU) (sklearn, 2015), hybrid CPU/GPU mean-shift

[2] https://github.com/masqm/Faster-Mean-Shift

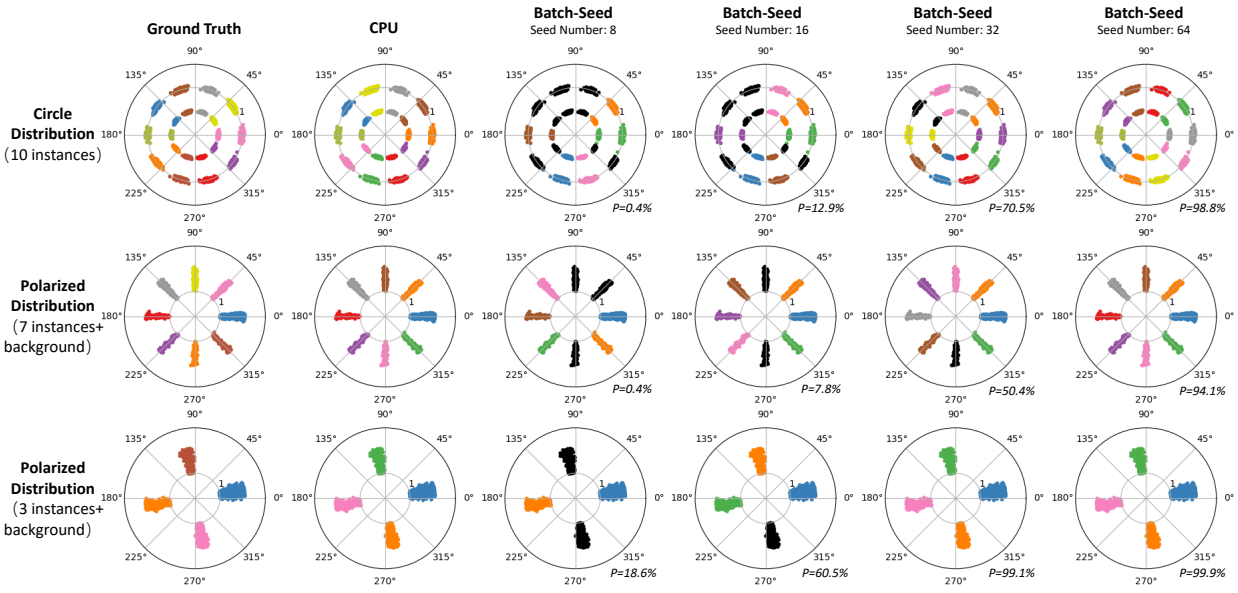


Fig. 4: This figure presents the simulation results using the proposed Faster Mean-shift algorithm. Each row indicates a distribution of the cosine embedding, while the columns represent the different number of seeds. In each subplot, different colors represent different clusters. Note that the blue points in the second and third rows simulate the image background, with more pixels. The black points indicate the failed clusters that are not captured by mean-shift. The theoretical probabilities of successful mean-shift clustering from OSOP are provided at the lower right corner of each subplot, which indicates the probability p that each cluster contains at least one seed (Eq. (14)).

(GPU_H) (Miaoqing et al., 2013), and fully parallel mean-shift (GPU_P) (Fang et al., 2019).

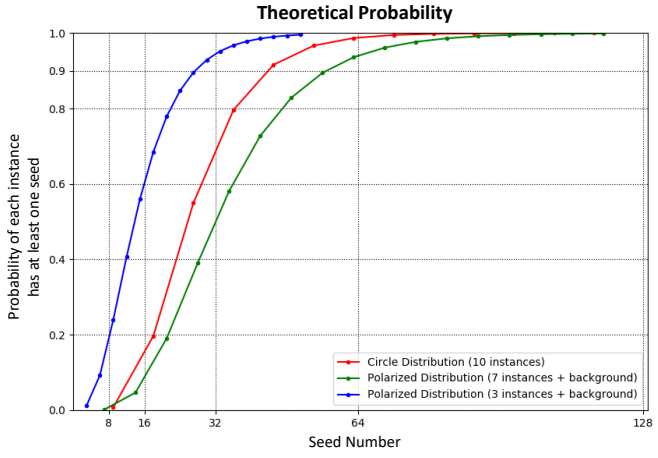


Fig. 5: This figure presents the theoretical probability of correct clustering (each cluster has been assigned at least one seed) with different numbers of seeds for simulation. The three theoretical curves match the three simulation experiments in Figure. 4.

For the simulations (in Figure 4), the h parameter was set to 0.1. We set different numbers of seeds for all experiments to evaluate the effects of such a hyper parameter. For the speed and memory test (in Table 1, Figure 6 and 7), we recorded the computational time and GPU memory cost on each data-set using NVIDIA Management Library (NVML) (NVIDIA, 2020). The peak GPU memory usage was reported as the GPU memory cost. To ensure robustness, we repeated the above experiment five times and reported the average measurements as the final results.

4.2.3. Results

The clustering results are presented in Figure 4, where the black color clusters indicate the failures. Figure 5 shows the theoretical probability of successful instance segmentation across different seed numbers, which matches the simulation results in Figure 4. Meanwhile, we also evaluated the time cost (Table 1 and Figure 6) and GPU memory cost (Figure 7) of the proposed Faster Mean-shift algorithm, compared with the baseline methods. In Figure 4), our algorithm correctly clustered the points in circle distributions when the number of seed is 64, which matched the theoretical curves in Figure 5. The remaining two simulations all matched their theoretical curves as well. The simulations demonstrated the number of seeds matched our theoretical derivations.

The costs of computational time and GPU memory are presented in Table 1, Figure 6 and 7. Briefly, in Table 1 and Figure 6, we tested our algorithm with different sizes of vectors and clusters, compared with the CPU (sklearn, 2015), GPU_H (Miaoqing et al., 2013), and GPU_P (Fang et al., 2019). Figure 6 shows that as the number of data points increased, the computational time of the CPU version increased dramatically. Within GPU algorithms, when the dataset was 10K-20K, the baseline methods performed slightly better than our algorithm. However, our algorithm achieved better speed performance after 20K. Considering a small 256×256 image consists of more than 65K pixels, our algorithm has superior computational speed for image processing.

GPU memory consumption results are presented in Figure 7. The GPU memory overhead of GPU_H was the lowest. However, it yielded worse computational efficiency (Miaoqing et al., 2013). For GPU_P, as an opposite extreme case, which has large GPU consumption to achieve fast computational speed. Our

Table 1: **Quantitative Results of Simulation**

Method	Metric	Number of 2-D Vectors							
		1K	2K	5K	10K	20K	50K	100K	200K
CPU (sklearn, 2015; Payer et al., 2019)	Time Cost (s/frame)	**0.078**	**0.254**	**0.734**	2.439	6.429	27.901	121.725	446.319
	GPU Memory (MB)	-	-	-	-	-	-	-	-
GPU_H (Miaoqing et al., 2013)	Time Cost (s/frame)	1.059	1.076	1.122	**1.264**	**1.684**	2.832	11.213	32.622
	GPU Memory (MB)	216	216	216	216	216	218	238	236
GPU_P (Fang et al., 2019)	Time Cost (s/frame)	1.572	1.559	1.586	1.594	1.773	2.256	3.877	7.561
	GPU Memory (MB)	216	216	236	256	374	1174	3294	10156
Ours	Time Cost (s/frame)	1.624	1.618	1.664	1.698	1.747	**1.892**	**2.107**	**2.692**
	GPU Memory (MB)	236	236	256	296	374	608	1004	1766

*The CPU version of mean-shift does not use the GPU, so GPU memory is not available(-).

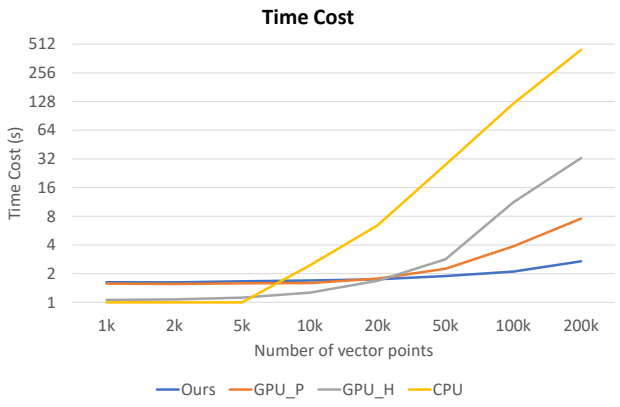


Fig. 6: This figure shows costs of computational time for different methods using simulations.

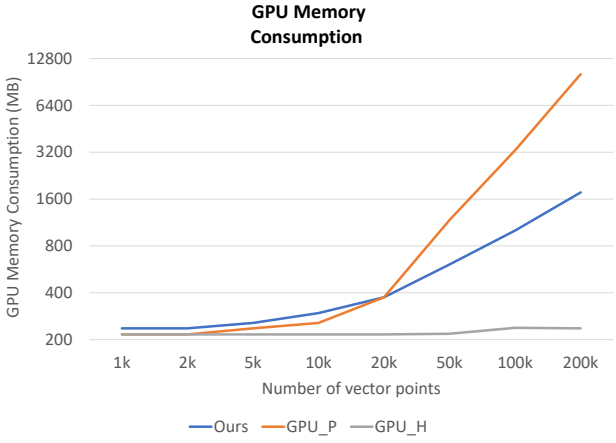


Fig. 7: This figure shows GPU memory consumption of different methods using simulations. Since the CPU version does not use GPU, only three GPU related methods are presented.

Faster Mean-shift achieved the highest computational speed as well as had less had less memory consumption than GPU_P. Therefore, for a standard single GPU card situation, the memory overhead of our algorithm was superior, especially for image processing.

The simulations demonstrated the superior computational speed of our proposed method as well as the efficient GPU memory consumption.

4.3. Empirical Validation

In order to evaluate the performance of the Faster Mean-shift algorithm on the real-world cell image instance segmentation and tracking, we applied our method on four cohorts from the ISBI cell tracking challenge (Ulman et al., 2017).

4.3.1. Data

In this testing, we used microscope video sequences from the ISBI cell tracking challenge (Ulman et al., 2017). Four cell image datasets of different sizes, shapes, and textures were adopted to test the Faster Mean-shift clustering: (1) HeLa cells on a flat glass (DIC-C2DH-HeLa) with 512×512 pixels, (2) GFP-GOWT1 mouse stem cells (Fluo-N2DH-GOWT1) with 1024×1024 pixels, (3) Simulated nuclei of HL60 cells stained with Hoescht (Fluo-N2DH-SIM+) with 696×520 pixels, and (4) Glioblastoma-astrocytoma U373 cells on a polyacrylamide substrate (PhC-C2DH-U373), with 660×718 pixels. In addition, the testing results of remaining three data-sets are also shown in the Appendix B.

4.3.2. Design

The cell tracking and segmentation model proposed by Payer et al (Payer et al., 2019). with mean-shift clustering, was used as the benchmark. In the original model, the CPU version of mean-shift algorithm from sklearn (sklearn, 2015) was used to cluster the cosine embedding vectors generated by RSHN. We replaced the original CPU version mean-shift algorithm and used our proposed Faster Mean-shift algorithm for clus-

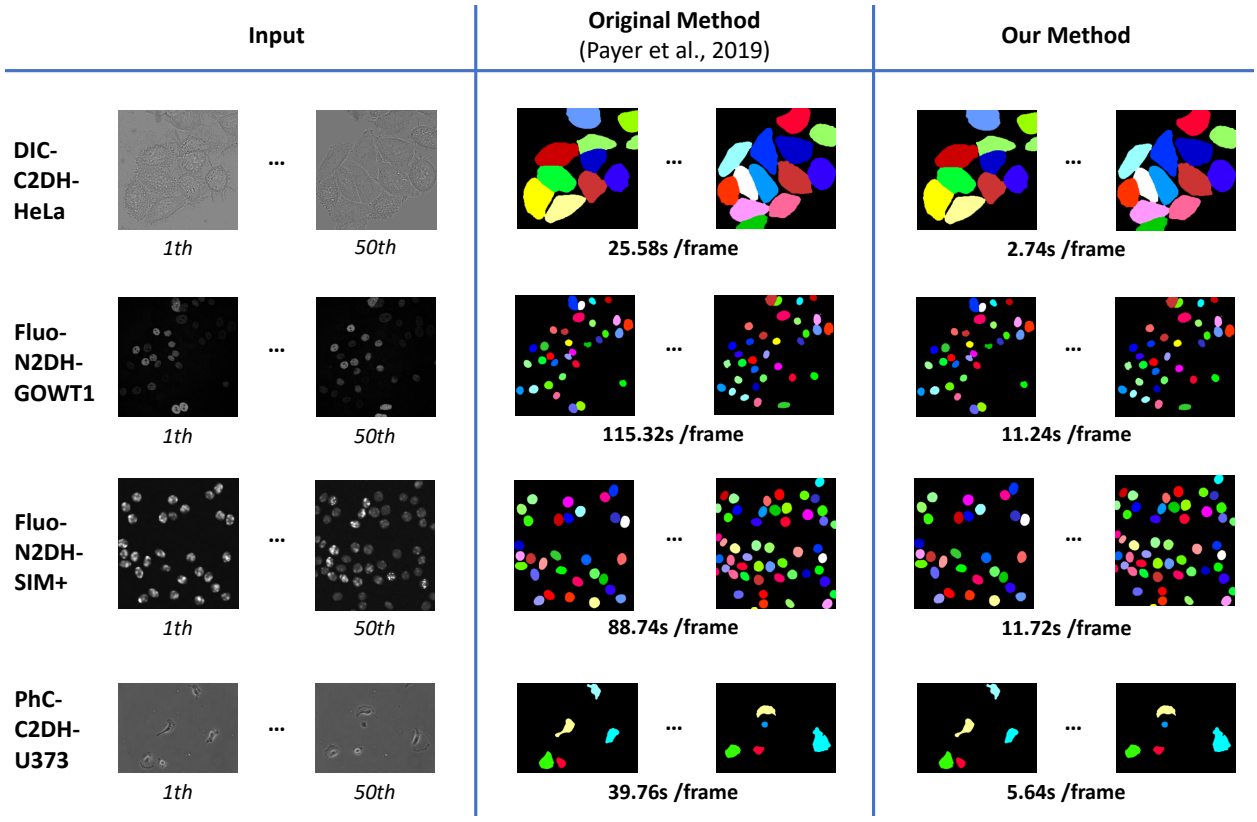


Fig. 8: This figure shows the qualitative instance segmentation and tracking results of the different methods. The left panel shows the input images. The middle panel shows the instance segmentation and tracking results of the baseline method. The right panel presents the results of the proposed method. The time costs per frame are also shown for different methods.

Table 2: **Quantitative Result of Empirical Validation**

Method	**Metric**	**Data-Set**			
		DIC-C2DH-HeLa	Fluo-N2DH-GOWT1	Fluo-N2DH-SIM+	PhC-C2DH-U373
Original(CPU) (Payer et al., 2019; sklearn, 2015)	Time Cost (s/frame)	25.58	115.32	88.74	39.76
	GPU Memory (GB)	5.92	5.92	5.92	5.92
GPU_H (Miaoqing et al., 2013)	Time Cost (s/frame)	9.83	37.54	18.60	10.81
	GPU Memory (GB)	6.16	6.16	6.16	6.16
GPU_P (Fang et al., 2019)	Time Cost (s/frame)	-	-	-	-
	GPU Memory (GB)	overflow	overflow	overflow	overflow
Ours	Time Cost (s/frame)	**2.74**	**11.24**	**11.72**	**5.64**
	GPU Memory (GB)	8.38	7.24	6.95	6.82

*5.92GB is the minimal GPU memory consumption for the deep learning based feature extraction.

*Due to the GPU overflow, the time-cost is not available(-)

tering. Moreover, we also replaced CPU version mean-shift with GPU_H and GPU_P (Ablation Studies). However, due to the limitation of GPU memory, when using GPU_P, the GPU card encountered a memory overflow. As a result, speed performance and GPU memory consumption are not available. In the experiments, h was set to 0.1 according to (Payer et al., 2019). The $N_{initial}$ was set to 128 based on Figure 5. During the testing phase, 50 frames from each of the above four data-sets were employed for testing. We recorded the resource consumption by repeating the algorithm three times and reported the average results.

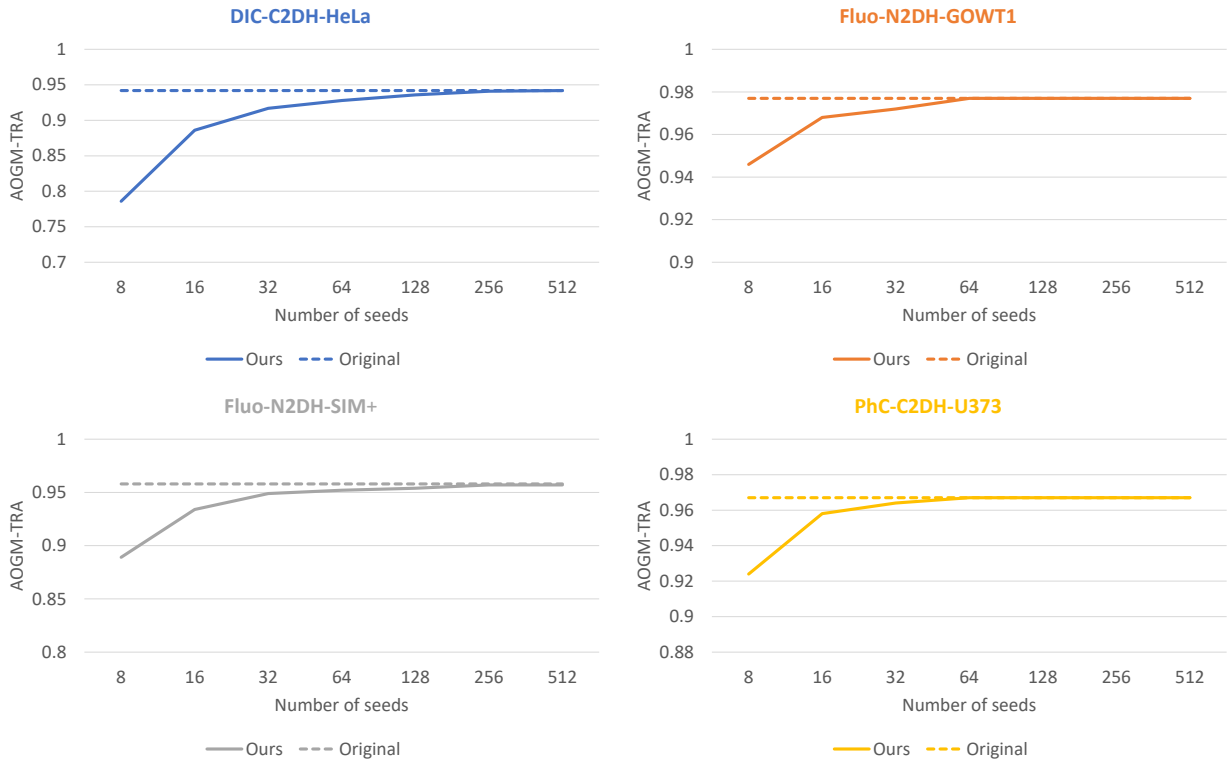


Fig. 9: This figure presents the standard TRA measurements (Ulman et al., 2017) for the Faster Mean-shift (solid lines) and upper bound (dashed lines) when using all pixels for clustering.

4.3.3. Results

The qualitative results were shown in Figure 8, and the computational speed results were shown in Table 2. For the cell image data-sets, our algorithm achieved a 7-10 times speedup compared with (Payer et al., 2019).

In Figure 9, the normalized Acyclic Oriented Graph Matching measure for tracking(AOGM-TRA)(Ulman et al., 2017),(Matula et al., 2015) was used as the accuracy metric. When the number of the seeds was higher than 256, the performance reached its upper bound limitation. The results demonstrated that the speed-up of our Faster Mean-shift did not sacrifice the accuracy.

Table 3: **Time-cost (per frame) for Different $N_{initial}$**

$N_{initial}$	=64	=128	=256
DIC-C2DH-HeLa	2.59s	2.74s	2.98s
Fluo-N2DH-GOWT1	12.18s	11.24s	11.26s
Fluo-N2DH-SIM+	11.24s	11.72s	11.52s
PhC-C2DH-U373	5.66s	5.64s	5.57s

Moreover, the time cost for different $N_{initial}$ are provided in Table 3. $N_{initial}$ will affect the number of iterations in the first few frames. However, the influence of $N_{initial}$ on the average computational time is small as shown in Table 3.

H will also affect the time-cost. In general, the required computational time per frame slightly increases with the larger H. And, the influence on average time-cost is shown in Table4.

Table 4: **Time-cost (per frame) for Different H**

H	=4	=8	=16
DIC-C2DH-HeLa	2.69s	2.74s	2.74s
Fluo-N2DH-GOWT1	11.02s	11.24s	11.30s
Fluo-N2DH-SIM+	11.36s	11.72s	11.81s
PhC-C2DH-U373	5.56s	5.64s	5.67s

5. Discussion

5.1. Performance Analyze

The computational time cost per frame of different methods on different cohorts was shown in Table 2. For the cell image data-sets, our algorithm achieved the best computational efficiency across four cohorts. It is worth to mentioning that the speed improvement of total computational time in empirical validation is less than the simulation. The reason is that the time cost of an empirical validation consists of (1) the forward path of deep learning, and (2) the mean-shift clustering. However, in simulation, the time cost only contains the mean-shift clustering. As the forward deep learning path is included for the real data-set, the speed improvement of the total computational time in simulation is larger than the real data-set. However, considering that the RSHN required a considerable amount of time to generate cosine embedding, our algorithm achieved great improvement.

The memory consumption using different methods on different cohorts was shown in Table 2. Our algorithm achieved much

lower total GPU memory costs with GPU_P, which was acceptable for a prevalent GPU card (e.g., GeForce RTX 2080 Ti with 11 GB GPU memory). Note that the 5.92GB GPU memory was required for all methods as this is the minimally required GPU memory for running RSHN feature extraction using TensorFlow.

5.2. Overlapping and Dense Cells

As the cell tracking challenge data do not consist of a large number of heavily overlapped cells, we performed a simulation as shown in Figure A.10 in Appendix A to evaluate the performance of the proposed method at highly overlapped scenarios. From the results, the required theoretical $N_{\min}$ (from Eq. (19)) increased with higher overlapping ratios. The proposed method achieved accurate segmentation results using the same settings ($L = 2$ and $H = 8$) as other experiences across this study. The simulation shows that the proposed Faster Mean-shift algorithm is able to cluster objects with different overlapping ratios.

Other issue is that the instance objects might not be uniformly distributed across the whole image, especially for high-resolution images. To tackle that challenge, the RSHN method has considered the non-uniform and dense distribution scenarios with patch-based design (Payer et al., 2019), where the high-resolution images are split to multiple 256×256 image patches. Then the patches are processed and aggregated to the original resolution. The performance of the proposed method on dense object quantification is presented in PhC-C2DL-PSC dataset in Appendix C.

5.3. Limitations

There are several limitations of current cosine-embedding based instance cell segmentation and tracking as well as the proposed Faster Mean-shift. First, the higher image resolution images, especially in 3D imaging, might lead to significantly larger GPU memory consumption, and may not fit the current hardware. Second, one major limitation of the proposed method is that the method is designed for 2D cell imaging. However, the capability of processing 3D cell images would be critical, especially with the rapid development of 3D microscopy imaging. Therefore, a valuable future direction would be to extend the proposed method from 2D to 3D. Third, even though the Faster Mean-shift accelerates the inference stage by a large margin, the speed is still not at a real time scale. Moreover, the cell tracking challenge data do not consist of a large number of heavily overlapped cells. However, it is important to compare the performance of the proposed method with highly overlapped objects versus minimal overlapping scenarios. Herein, we performed a simulation as shown in Fig. A.10 in Appendix A.

6. Conclusion

In this study, we proposed a Faster Mean-shift algorithm for tackling the bottleneck of the state-of-the-art cosine embedding based cell instance segmentation and tracking. Compared with previous GPU-based mean-shift algorithms, our Faster Mean-shift method achieved better computational speeds, with acceptable memory consumption for a single ordinary GPU card. Using Faster Mean-shift, the processing speed for each frame was accelerated by 7-10 times compared to the state-of-the-art embedding based cell instance segmentation and tracking algorithm. As many recent studies have demonstrated the significant advantages and superior accuracy performance of embedding based methods, this algorithm provides a plug-and-play model, which is adapted for any pixel embedding based clustering inference.

7. Acknowledgements

This study is supported by NSF Career Award 1452485 (Landman).

References

Allen, W.E., Zicha, D., Ridley, A.J., Jones, G.E., 1998. A role for cdc42 in macrophage chemotaxis. The Journal of cell biology 141, 1147–1157.

Ballas, N., Yao, L., Pal, C., Courville, A., 2015. Delving deeper into convolutional networks for learning video representations. arXiv preprint arXiv:1511.06432 .

Bhimani, J., Leeser, M., Mi, N., 2015. Accelerating k-means clustering with parallel implementations and gpu computing, in: 2015 IEEE High Performance Extreme Computing Conference (HPEC), IEEE. pp. 1–6.

Bulte, J.W., 2009. In vivo mri cell tracking: clinical studies. American Journal of Roentgenology 193, 314–325.

Campello, R.J., Moulavi, D., Zimek, A., Sander, J., 2015. Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Transactions on Knowledge Discovery from Data (TKDD) 10, 1–51.

Chalela, M., Sillero, E., Pereyra, L., García, M.A., Cabral, J.B., Lares, M., Merchán, M., 2019. Grispy: A python package for fixed-radius nearest neighbors search. arXiv preprint arXiv:1912.09585 .

Chavez, L.R., 2001. Covering immigration: Popular images and the politics of the nation. University of California Press Berkeley.

Chen, H., Qi, X., Yu, L., Dou, Q., Qin, J., Heng, P.A., 2017. Dcan: Deep contour-aware networks for object instance segmentation from histology images. Medical image analysis 36, 135–146.

Chen, L., Strauch, M., Merhof, D., 2019. Instance segmentation of biomedical images with an object-aware embedding learned with local constraints, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer. pp. 451–459.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 .

Comaniciu, D., Meer, P., 2002. Mean shift: A robust approach toward feature space analysis. IEEE Transactions on pattern analysis and machine intelligence 24, 603–619.

Condeelis, J., Pollard, J.W., 2006. Macrophages: obligate partners for tumor cell migration, invasion, and metastasis. Cell 124, 263–266.

Czirók, A., Schlett, K., Madarász, E., Vicsek, T., 1998. Exponential distribution of locomotion activity in cell cultures. Physical Review Letters 81, 3038.

Debeir, O., Van Ham, P., Kiss, R., Decaestecker, C., 2005. Tracking of migrating cells under phase-contrast video microscopy with combined mean-shift processes. IEEE transactions on medical imaging 24, 697–711.

Fang, H., Chen, Y., Li, L., Zhou, J., Tao, J., Tan, X., Fan, G., 2019. Implementation of the parallel mean shift-based image segmentation algorithm on a gpu cluster. International Journal of Digital Earth 12, 328–353.

Fukunaga, K., Hostetler, L., 1975. The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Transactions on information theory 21, 32–40.

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 580–587.

Graham, S., Chen, H., Gamper, J., Dou, Q., Heng, P.A., Snead, D., Tsang, Y.W., Rajpoot, N., 2019. Mild-net: minimal information loss dilated network for gland instance segmentation in colon histology images. Medical image analysis 52, 199–211.

Guo, Q., Chang, X., Chu, H., 2007. Mean-shift of variable window based on the epanechnikov kernel, in: 2007 International Conference on Mechatronics and Automation, IEEE. pp. 2314–2319.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural computation 9, 1735–1780.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

Kulikov, V., Lempitsky, V., 2020. Instance segmentation of biological images using harmonic embeddings, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 3843–3851.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86, 2278–2324.

Li, M., Chen, X., Li, X., Ma, B., Vitányi, P.M., 2004. The similarity metric. IEEE transactions on Information Theory 50, 3250–3264.

Matula, P., Maška, M., Sorokin, D.V., Matula, P., Ortiz-de Solórzano, C., Kozubek, M., 2015. Cell tracking accuracy measurement based on comparison of acyclic oriented graphs. PloS one 10, e0144959.

Miaoqing, H., Men, L., Lai, C., 2013. Accelerating mean shift segmentation algorithm on hybrid cpu/gpu platforms, in: Modern Accelerator Technologies for Geographic Information Science. Springer, pp. 157–166.

Microsoft, 2020. Azure nc-series. `https://docs.microsoft.com/en-us/azure/virtual-machines/nc-series`.

Montell, D.J., 2008. Morphogenetic cell movements: diversity from modular mechanical properties. Science 322, 1502–1505.

Newell, A., Yang, K., Deng, J., 2016. Stacked hourglass networks for human pose estimation, in: European conference on computer vision, Springer. pp. 483–499.

NVIDIA, 2015. Nvidia, v. (2013). tesla k20 gpu accelerator board specification. `https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/Tesla-K80-BoardSpec-07317-001-v05.pdf`.

NVIDIA, 2020. Nvidia management library (nvml). `https://developer.nvidia.com/nvidia-management-library-nvml`.

Payer, C., Štern, D., Feiner, M., Bischof, H., Urschler, M., 2019. Segmenting and tracking cell instances with cosine embeddings and recurrent hourglass networks. Medical image analysis 57, 106–119.

Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer. pp. 234–241.

Rosenthal, A., Mork, P., Li, M.H., Stanford, J., Koester, D., Reynolds, P., 2010. Cloud computing: a new business paradigm for biomedical information sharing. Journal of biomedical informatics 43, 342–353.

Senoussaoui, M., Kenny, P., Stafylakis, T., Dumouchel, P., 2013. A study of the cosine distance-based mean shift for telephone speech diarization. IEEE/ACM Transactions on Audio, Speech, and Language Processing 22, 217–227.

sklearn, 2015. Scikit-learn machine learning in python. `https://scikit-learn.org/stable/index.html`.

Sutton, E.J., Henning, T.D., Pichler, B.J., Bremer, C., Daldrup-Link, H.E., 2008. Cell tracking with optical imaging. European radiology 18, 2021–2032.

Ulman, V., Maška, M., Magnusson, K.E., Ronneberger, O., Haubold, C., Harder, N., Matula, P., Matula, P., Svoboda, D., Radojevic, M., et al., 2017. An objective comparison of cell-tracking algorithms. Nature methods 14, 1141–1152.

Webb, S.E., Pollard, J.W., Jones, G.E., 1996. Direct observation and quantification of macrophage chemoattraction to the growth factor csf-1. Journal of Cell Science 109, 793–803.

Xiao, C., Liu, M., 2010. Efficient mean-shift clustering using gaussian kd-tree, in: Computer Graphics Forum, Wiley Online Library. pp. 2065–2073.

Zhao, M., Chang, C.H., Xie, W., Xie, Z., Hu, J., 2020. Cloud shape classification system based on multi-channel cnn and improved fdm. IEEE Access 8, 44111–44124.

Zimmer, C., Zhang, B., Dufour, A., Thébaud, A., Berlemont, S., Meas-Yedid, V., Marin, J.C., 2006. On the digital trail of mobile cells. IEEE Signal Processing Magazine 23, 54–62.

Appendix A. Overlapping Simulation

A simulation is provided to evaluate the theoretical numbers of seeds $N_{\min}$ based on the ratio between foreground and the entire image r using (19) with probability P = 0.95. The resolution of the simulated data is 1000×1000×18, where 1000×1000 is the total numbers of feature vectors for an image and 18 is the dimension of cosine embedding for each vector. The simulated data are presented as color images for visualization in Fig.A.10, where each simulated data contains eight foreground circle objects (diameter d = 250 pixels) with different cosine embeddings. The different levels of overlapping are simulated by controlling the intersection between circles. The instance segmentation results using the theoretical $N_{\min}$ with different r values are presented in Figure A.10. From the results, the required $N_{\min}$ increased with a higher overlapping ratio. The computational times of different simulations are presented. The results showed that the default setting settings ($L = 2$ and $H = 8$) achieved the correct segmentation results. Note that this simulation did not include random noise, which is different from the ones in Figure 4 Since this simulation is to evaluate the effects of overlapping, we avoid the potential entangled impacts from noise.

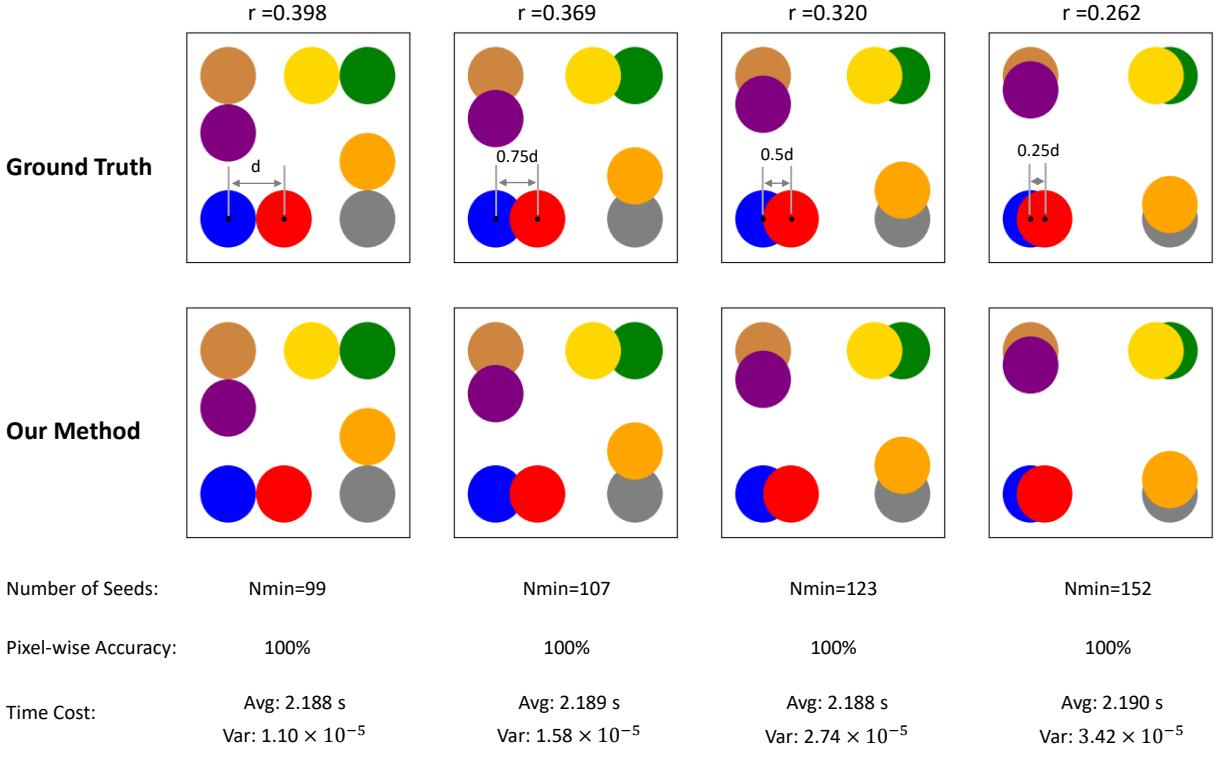


Fig. A.10: This figure shows the simulation on different ratios of overlapping. The first row shows the ground truth of simulation. Different colors represent different cosine embeddings. The second row shows the results of the proposed method. The computational time of different simulations are also presented.

Appendix B. Three Supplementary Data-sets

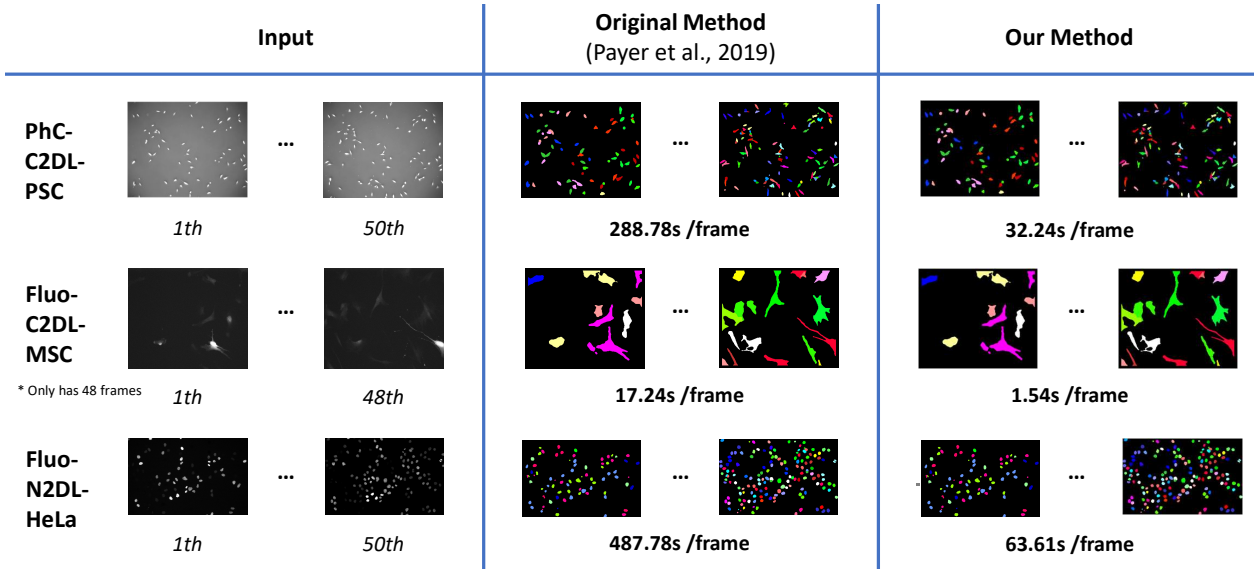


Fig. B.11: This figure shows the qualitative instance segmentation and tracking results of three data-sets from Cell Tracking Challenge. The left panel shows the input video frames. The middle panel shows the instance segmentation and tracking results using the baseline method. The right panel presents the results using the proposed method. The time costs per frame are also shown for different methods.

Table B.5: **Quantitative Result of Empirical Validation**

Method	Metric	Data-Set		
		PhC-C2DL-PSC	Fluo-C2DL-MSC	Fluo-N2DL-HeLa
Original(CPU) (Payer et al., 2019; sklearn, 2015)	Time Cost (s/frame)	288.78	17.24	487.78
	GPU Memory (GB)	5.92	5.92	5.92
GPU_H (Miaoqing et al., 2013)	Time Cost (s/frame)	89.94	7.89	121.45
	GPU Memory (GB)	6.16	6.16	6.16
GPU_P (Fang et al., 2019)	Time Cost (s/frame)	-	-	-
	GPU Memory (GB)	overflow	overflow	overflow
Ours	Time Cost (s/frame)	**32.24s**	**1.54**	**63.61**
	GPU Memory (GB)	7.49	6.21	8.58

*5.92GB is the minimal GPU memory consumption for the deep learning based feature extraction.

*Due to the GPU overflow, the time-cost is not available(-)

Appendix C. Dense Cells Segmentation

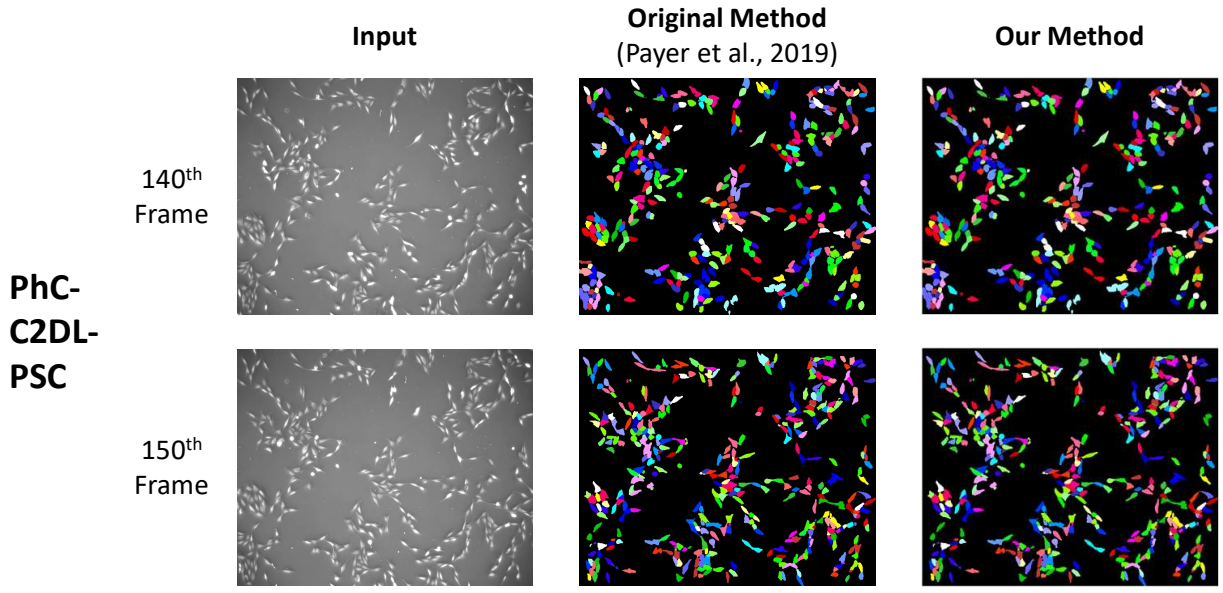
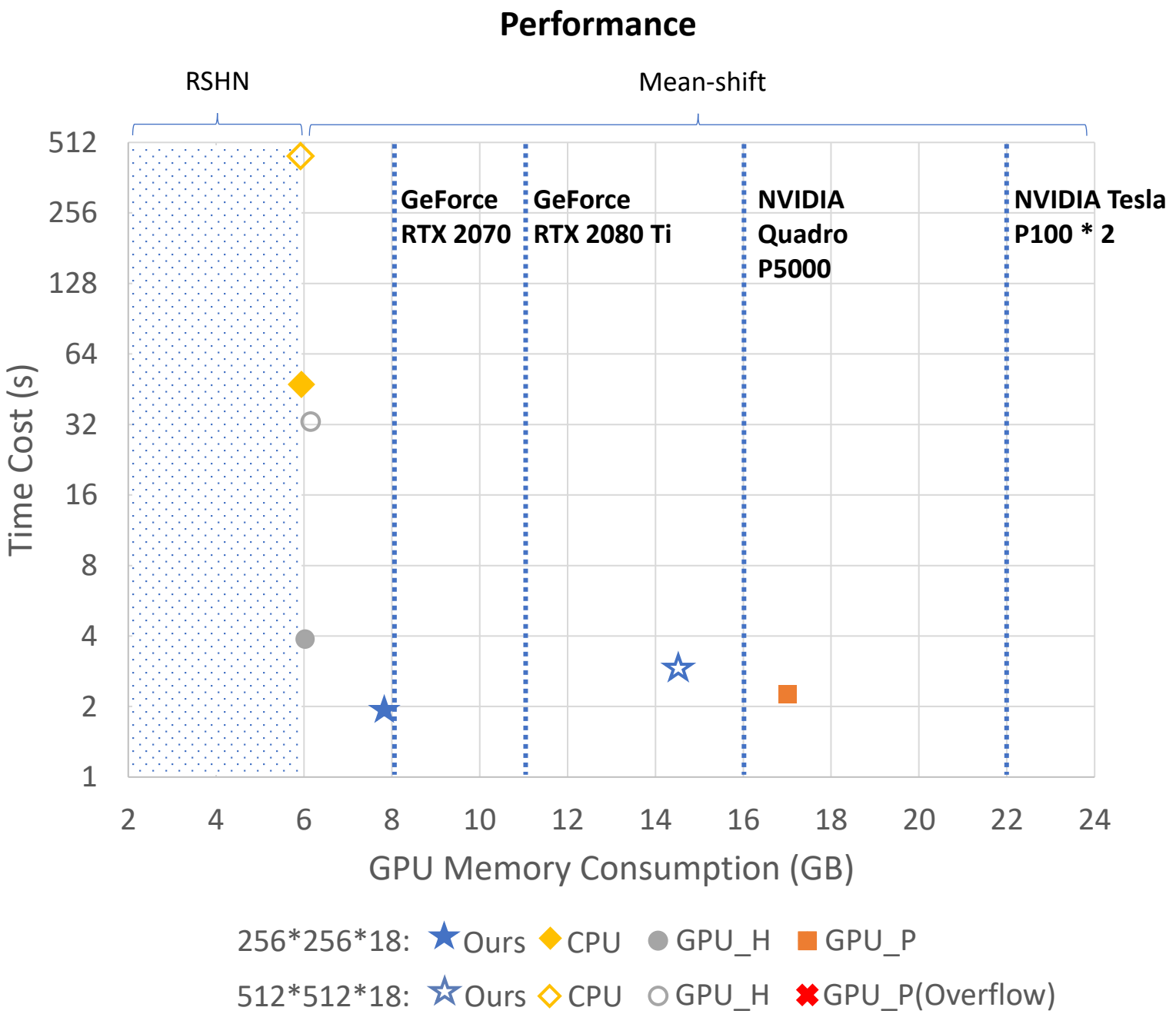


Fig. C.12: This figure shows the qualitative results of dense object segmentation.


Performance
RSHN
Mean-shift
GeForce RTX 2070
GeForce RTX 2080 Ti
NVIDIA Quadro P5000
NVIDIA Tesla P100 * 2
Time Cost (s)
512
256
128
64
32
16
8
4
2
1
2
4
6
8
10
12
14
16
18
20
22
24
GPU Memory Consumption (GB)
256*256*18: Ours CPU GPU_H GPU_P
512*512*18: Ours CPU GPU_H GPU_P(Overflow)